VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky

BAKALÁŘSKÁ PRÁCE

2011 **Michal Rečka**

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Absolvování individuální odborné praxe
Individual Professional Practice in the Company

2011 **Michal Rečka**

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Michal Rečka**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Absolvování individuální odborné praxe
Individual Professional Practice in the Company

Zásady pro vypracování:

1. Student vykoná individuální praxi ve firmě: CID International, a.s.
2. Struktura závěrečné zprávy:
   a) Popis odborného zaměření firmy, u které student vykonal odbornou praxi a popis pracovního zařazení studenta.
   b) Seznam úkolů zadaných studentovi v průběhu odborné praxe s vyjádřením jejich časové náročnosti.
   c) Zvolený postup řešení zadaných úkolů.
   d) Teoretické a praktické znalosti a dovednosti získané v průběhu studia uplatněné studentem v průběhu odborné praxe.
   e) Znalosti či dovednosti scházející studentovi v průběhu odborné praxe.
   f) Dosažené výsledky v průběhu odborné praxe a její celkové zhodnocení.

Seznam doporučené odborné literatury:

Podle pokynů konzultanta, který vede odbornou praxi studenta.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Tomáš Fabián**

Konzultant bakalářské práce: Ing. Petr Janoš

Datum zadání: 19.11.2010

Datum odevzdání: 06.05.2011

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 5. května 2011 .....

Abstrakt

Tato práce popisuje mou odbornou praxi ve firmě CID International a.s. Jsou zde uvedeny projekty, na kterých jsem pracoval a to jak ty, které jsem vytvářel od začátku, tak ty, které jsem přebral po jiných programátorech. Dva projekty, které považuji za vývojově složitější, jsou rozvedeny do větších detailů i s průběhem řešení. U všech projektů, ve kterých se daly získat nějaké měřitelné výsledky, jsou výsledky uvedeny.

Klíčová slova

CID International a.s., odborná praxe, CIDControl, modul, AutoComplete, CLR procedura

Abstract

This thesis deals with my professional praxis in CID International company. In this work I described all the projects I had been working on during this praxis. Especially I focused on two specific and more elaborate projects. Finally I tried to summarize my workflow including the results I have reached.

Key Words

CID International a.s., Individual Professional Practice in the Company, CIDControl, module, AutoComplete, CLR procedure

Seznam použitých symbolů a zkratek

CAB	Composite UI Application Block
CID	CID International a.s.
CRM	Customer Relationship Management
DB	Databáze
GUI	Graphical User Interface / Grafické uživatelské rozhraní
IS	Informační systém
LINQ	Language Integrated Query
OKD	Ostravsko-Karvinské doly
ORM	Objektově relační mapování
SCSF	Smart Client Software Factory
SOA	Service Oriented Architecture
SQL	Structured Query Language
TFS	Team Foundation Server
VRT	Vedoucí realizačního týmu
VS2010	Microsoft Visual Studio 2010
WCF	Windows Communication Foundation

Obsah

1. Úvod

Svou odbornou praxi jsem absolvoval u společnosti CID International a.s. v pozici *programátor externista*. V následujícím textu bude zmíněno něco o firmě samotné, stručný popis nástrojů, se kterými jsem pracoval, detailnější popis mých projektů a na závěr subjektivní zhodnocení odborné praxe.

2. Proč odborná praxe místo bakalářské práce

Důvodů, proč jsem se rozhodl absolvovat odbornou praxi, bylo několik. Nejdůležitějším faktorem byla potřeba finančního zázemí, které jsem jako student potřeboval a stále potřebuji. Hned za tím následovala potřeba praxe v oboru, kterou bych mohl využít při hledání zaměstnání a která by mě obohatila o věci, které se ve škole nenaučím. Není tajemstvím, že teorie a praxe se může diametrálně lišit, a proto jsem chtěl nahlédnout na studované téma i z jiného, úhlu pohledu, než je ten akademický. Jako plus vidím fakt, že odborná praxe slouží jako náhrada za tradiční bakalářskou práci, a to především z důvodu efektivního využití času.

3. CID International a.s.

3.1. Seznámení

„CID International a.s. je součástí obchodní skupiny Oltis Group a.s. CID vyvíjí informační systémy pro dopravu, spedici, skladování a CRM.“ [3] Společnost působí na trhu již od roku 1996 a dlouhodobě si udržuje i ocenění Microsoft Gold Certified Partner[1]. Svou odbornou praxi jsem absolvoval na pozici programátor externista, která je objasněna níže. Byl jsem členem realizačního týmu LOGI, ale často jsem pracoval i pro tým LORI.

3.2. Vnitřní uspořádání společnosti

3.2.1 Analytici

Základním kamenem jsou tzv. analytici. Analytik konzultuje se zákazníky jejich požadavky, následně na základě informací získaných od zákazníka zpracovává analýzu, kterou předává programátorům. Na této analýze pracuje až do ukončení projektu a po celou dobu vývoje upřesňuje programátorům informace, které jsou zapotřebí. Než je aplikace předána zákazníkovi, je analytik zodpovědný i za maximální možné otestování aplikace a v případě nalezené chyby aplikaci vrací programátorům. Po nasazení poskytují analytici technickou podporu zákazníkům, konzultují požadavky na změny a reklamace a odhadují přibližnou dobu implementace případných opatření.

[1] Více informací na http://www.cid.cz/clanek.m?Id=145 [4]

3.2.2. Programátoři

Programátoři podle analýzy a informací od analytiků vytvářejí výslednou aplikaci. V průběhu vývoje pak vydávají a dokumentují různé verze aplikačních částí, na kterých pracují. Testují nedostatky na úrovni kódů a snaží se optimalizovat kód pro maximální rychlost a spolehlivost. Speciální pozicí je **programátor externista**. Jediný rozdíl od klasického programátora je ten, že externisti mají zcela volnou pracovní dobu (co se týče počtu odpracovaných hodin měsíčně) a pracují na dohodu o provedení pracovní činnosti.

3.2.3. Obchodníci

Jejich prací je především komunikace se zákazníky, hledání nových zákazníků a prodávání produktů společnosti. To zahrnuje i komunikaci s vedoucími realizačních týmů a s ředitelem samotným, se kterými konzultují možnosti a podmínky obchodu.

3.2.4. Realizační týmy

CID vyvíjí IS pro různé cílové skupiny, podle kterých se také rozlišují i realizační týmy:

Název týmu:	Specializace týmu:
CARGI	Železniční spedice
COLLI	Balíková distribuce a sběrná služba
CSV	Centrální systém výluk
KONTI	Kontejnerová překladiště
LOGI	Správa logistických, výrobních a obchodních skladů
LORI	Osobní a nákladní automobilová doprava
MARKET	CRM systém pro logistické firmy
Weby	Tvorba webů k našim produktům

Tabulka 1: Realizační týmy CID International a.s.

Realizační týmy mají většinou 1-5 analytiků a 1-4 programátory či programátory externisty. Každý realizační tým vede minimálně 1 VRT.

3.2.5. Vedoucí realizačního týmu

Vedoucí realizačního týmu, nebo také „vrti“ jsou spolu s ředitelem mozkem společnosti. Pravidelně pořádají porady, diskutují a řídí veškeré dění ve firmě, určují priority prací, rozdělují práci jednotlivým členům týmů a v neposlední řadě jsou sami také buď analytiky, nebo programátory.

3.2.6. Ředitel

Nejvyšší autoritou je ředitel. Ten řídí strategii a chod společnosti a dohlíží na dodržování všech interních předpisů a metodik.

3.3. Vývoj informačních systémů

CID vyvíjí informační systém na základě Smart Client Software Factory. Jedná se o princip softwarové továrny v prostředí .NET pro Windows Forms. Většinou jde o vývoj v jazyce C# a jako datová základna slouží databáze MS SQL Server. Počátkem roku 2011 byly převedeny všechny důležité a dále podporované součásti do .NET 4.0 a práce ve starších verzích .NET jsou již pouze na základě technické podpory a oprav.

3.3.1. Smart Client Software Factory (SCSF)

„SCSF je softwarová továrna, která slouží k vývoji tzv. chytrých klientů. (...) Je založena na Composite UI Application Blocku (dále jen CAB). Jde o aplikační blok vytvořený skupinou Microsoft patterns & practices. Ta se věnuje návrhovým vzorům a efektivním praktikám a vydává doporučení pro vývoj aplikací. Doporučení jsou vydávána právě pomocí aplikačních bloků. Asi nejznámějšími produkty této skupiny jsou Enterprise Library, Composite WPF, Unity Application Block a právě Composite UI Application Block.“ [2] Více informací lze čerpat z knihy Davida S, PLATTA *Programming Microsoft® Composite UI Application Block and Smart Client Software Factory*. [6]

3.3.2. Průzkumník (shell)

Průzkumník je hlavní aplikace, která zahrnuje hlavní GUI informačního systému a slouží jako vnější kontejner pro všechny části CAB aplikace. Jinými slovy spouští se jako první a hostuje všechny ostatní části architektury SCSF. Moduly a services lze do průzkumníka načíst patřičnou konfigurací v XML souboru (reference nejsou předem známy, vše je propojeno tzv. volnou vazbou).

3.3.3. Služby (Services)

Na shell je napojena SOA (Service Oriented Architecture). Jde o služby, které obsahují logiku společnou pro celou aplikaci. Vytvářejí se teprve, když jsou potřeba, ale po jejich vytvoření již setrvávají, dokud nejsou explicitně uvolněny, nebo dokud aplikace neskončí. Typicky se do services dávají funkce jako například autentizace, autorizace, barvení controlů, validace, sdílení datových struktur apod.

3.3.4. Components / Controls

Některé prvky view jsou využívány opakovaně na mnoha místech a jejich složitost není triviální. Aby byl vývoj efektivnější, jsou tyto prvky vytvářeny jako UserControl či Component. To pak umožňuje „znovu použitelnost“ komponent, výrazně šetří čas vývojářů a především případné změny se provádějí pouze na jednom místě. Často využíváme komponenty od společnosti Syncfusion, která se na jejich vývoj specializuje. Dobrými příklady jsou například UserControly pro práci s datem a časem, pro práci s adresami, GPS souřadnicemi, aj.

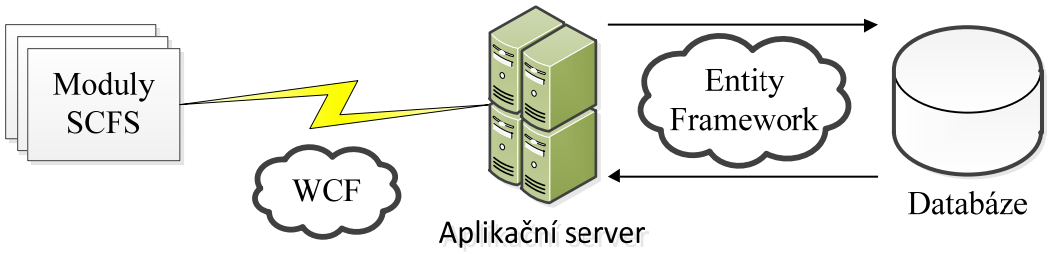
3.3.5. Moduly

Všechny výše zmíněné součásti IS by samy o sobě byly k ničemu, kdyby neexistovaly moduly. Modul je ucelená část logiky, která je zaměřená na určité konkrétní úlohy. Většinou je modul assembly, která obsahuje nějaké view, se kterým pak pracuje uživatel. Moduly jsou navzájem nezávislé, přesto jsou ale schopny spolu navzájem komunikovat přes volné vazby pomocí tzv. Event Publications a Event Subscriptions (ty v podstatě odpovídají klasickým událostem v .NET, ale nemají na sebe přímou referenci). Pokud modul obsahuje view, pak využívá návrhový vzor Model-View-Presenter, který odděluje datový zdroj (v tomto případě z pravidla databáze), aplikační logiku a prezentaci dat v grafickém uživatelském rozhraní. Příkladem modulu je např. modul pro vedení informací o zásilkách, modul pro výpočet diet pro zaměstnance, modul pro generování jízdních řádů a dohledávání spojů autobusů, apod.

3.3.6. Přístup k databázi

Informační systém částečně počítá s budoucím provozem ve 3 vrstvé architektuře. To znamená, že aplikace si řekne o data aplikačnímu serveru, ten požadavek vyhodnotí a dle potřeby si stáhne potřebné záznamy z databáze. Zpátky následně aplikační server vrací data přemapovaná na požadované objekty.

V tuto chvíli je ale aplikační server teprve ve vývoji a proto je druhá vrstva dočasně vypuštěna (systém přístupu k datům přes aplikační server vyžaduje velký zásah do stávající struktury, nicméně v nejbližší době by měl být nasazen k testování). Pro samotnou komunikaci s databází MS SQL Server je tedy zatím nejčastěji používáno objektově relační mapování LINQ to Entity a následně LINQ to SQL a ADO .NET. V praxi jsme ale zjistili, že LINQ to Entity má nezanedbatelný negativní vliv na výkon a nyní je testována alternativa - Entity Framework, který by měl být podle posledních testů rychlejší.



Obrázek 1: Schéma připravované komunikace s databází přes aplikační server

4. Moje práce

4.1. Software, se kterým jsem pracoval

4.1.1. Visual Studio 2010 Premium

Jakožto programátor jsem pracoval drtivou většinu času ve Visual Studiu 2010. Visual Studio je vývojové prostředí pro technologii .NET. Obsahuje téměř vše potřebné pro pohodlný vývoj aplikací od textového editoru s barevným formátováním, přes debugger až po monitorovací utility. Osobně jsem si oblíbil i pár bezplatných rozšíření jako např. DPack, Productiviy Power Tools aj.

4.1.2. MS SQL Management Studio 2005

Nezbytný nástroj pro práci s databází MS SQL Server je Management Studio. Zahrnuje téměř veškeré nástroje pro práci s databází od zadávání SQL dotazů či T-SQL kódů, úpravy struktur tabulek, až po správu databází samotných. Skoro vše se dá nějakým způsobem „naklikat“. Za zmínku určitě stojí i rozšíření SQLPrompt od společnosti RedGate, jenž poskytuje intellisense (našeptávač) pro SQL dotazy.

4.1.3. EQATECH Profiler

Monitorovací nástroj, který umožňuje docela hezky kontrolovat čas běhu jednotlivých metod a díky tomu efektivně dohledávat místa problémů, které aplikaci brzdí. Po ukončení testované aplikace program vygeneruje odpovídající orientované grafy a tabulky s výsledky.

4.1.4. .NET Reflector

Utilita, která umí pomocí reflexe z přeložené assembly vygenerovat zpětně zdrojový kód. Ačkoli se tato funkcionalita zdá být zbytečná pro vlastní vývoj, pro mne osobně byla více než dobrým pomocníkem. .NET Reflector sehrál často klíčovou roli zejména při optimalizacích, kdy jsem hledal nejefektivnější metody a často jsem se musel dívat i dovnitř cizích assembly, ke kterým jsem zdrojové kódy neměl (např. samotné knihovny .NET prostředí).

4.1.5. MS Visual Source Safe a Team Foundation Server

Programy pro správu verzí souborů. Všechny projekty musí být napojeny na Source Safe databázi, která umožňuje ukládat jednotlivé verze souborů podle data uložení. Také umožňuje uzamknout soubory pro jednoho uživatele, tudíž se nemůže stát, že by jeden soubor byl souběžně editován více lidmi najednou. Pomocí Source Safe databáze lze pak získat verzi souboru z kteréhokoli dříve uloženého data, prohlížet změny mezi jednotlivými verzemi i s údaji, kdo změny provedl. Ukázka práce se Source Safe viz *Příloha 5: Ukázka práce se Source Safe (plugin TFS ve VS2010)*.

4.2. Zásady pro efektivní práci

Abych mohl pracovat efektivně a s nadhledem, musel jsem si vybudovat určitý systém zásad, které se snažím dodržovat při vyvíjení projektů. V podstatě dodržuji stejné standardy, jako jsou doporučeny v článku Brada Abramse *Internal Coding Guidelines* [1], ale je třeba uvést pár doplnění. Ve firmě je zažité maďarské pojmenovávání proměnných (např. szNazev, dtDatumMod, lStav, …), proto se jej snažím alespoň v názvech proměnných u public metod dodržovat. Stejně tak jsou zažity privátní proměnné s předponou _. Následující úpravy už jsou převážně z mé iniciativy.

4.2.1. Struktura kódu

Za jednu z nejužitečnějších pomůcek považuji #region bloky. Slouží ke skrývání a zobrazování vymezených částí kódu s příslušným titulkem a podstatně zlepšují přehlednost kódu. Snad každý kód (s výjimkou automaticky generovaného), který jsem vytvářel od začátku sám, má přesné pořadí #region bloků: delegáty (Delegates), události (Events), třídní private proměnné (Fields), vlastnosti (Properties), konstruktory (Constructors), public metody (Public Methods), internal a protected metody (Internal Methods), private metody (Private Methods) a na konec obsluhy událostí (Event Handlers). V závorkách jsou konkrétní názvy #region bloků, které byste v kódu našli. Uvnitř každého bloku jsou prvky seřazeny abecedně od a po z, s tím že písmeno CH beru jako C a H. Ukázka využití #region bloků pro strukturaci kódu viz *Příloha 4: Ukázka #REGION struktury kódu v CidControlAutoComplete*.

4.2.2. Komentáře

Vše, co jde a co používám přímo v kódu, opatřuji XML komentáři. Má to několik výhod. Za prvé se mi kdekoli v kódu po najetí ukazatelem myši na takto „otagovaný“ prvek zobrazí informace, které jsem do tagů zadal. Za druhé když assembly zkompiluji, vytvoří se XML dokumentace, kterou mohou použít následně i jiní. Existuje však jedna výjimka, kterou jsou obsluhy událostí. Ty bývají generovány ve tvaru ‚nazevPrvku_nazevAkce‘ a mají podle mne dostatečně výstižný název na to, aby člověk věděl, se kterým prvkem metoda souvisí a kdy se asi metoda volá.

Dále se snažím přinejmenším na složitých místech psát i in-line komentáře s vysvětlivkami složitějších kroků. Měl jsem tu čest přebírat po jiných pár úplně neokomentovaných projektů a musím přiznat, že zorientovat se v takovémto kódu je téměř obtížnější, než jej celý napsat znovu.

4.3. Projekty

4.3.1. Seznam projektů a jejich časová náročnost

Projekt	Čas. náročnost	Cíl	Stav
_scbDoprProstredek	237 hodin	vytvořit	hotovo
_scbAdmSkladovatel	109 hodin	vytvořit	hotovo
_scbAdmUkladatel	116 hodin	převzít, rozšířit	hotovo

_scbZasilKalk	43 hodin	vytvořit	hotovo
_anPlaPtr	80 hodin	převzít, rozšířit	hotovo
_anExpImpTXT	21 hodin	vytvořit	hotovo
_anIsopcXml	44 hodin	vytvořit	hotovo
_clrTurnusBitoveMapy	58 hodin	vytvořit	hotovo
CIDControlAutoComplete	69 hodin	vytvořit	hotovo
CIDControlFirma	87 hodin	vytvořit	hotovo
CIDControlAdresa	80 hodin	převzít, rozšířit	hotovo
Paměťový objekt aplikačního serveru	2-3 měsíce	vytvořit	probíhá

Tabulka 2: Přehled projektů, jejich časová náročnost a stav

4.3.2. CLR procedura pro Generování bitových map

V osobní dopravě jezdí řidiči tzv. turnusy. Turnusy mohou být definovány několika způsoby. Čísly 1-7 se označují turnusy jednotlivých dní, ‚X' znamená pouze pracovní dny – tedy 1-5. Pokud je turnus označen znaménkem ‚+', jede pouze ve státní svátky, v opačném případě ve státní svátky nejede. Rovněž se občas musí brát v potaz školní vyučování. ‚S' značí turnusy, které jezdí pouze ve školní dny, ‚P' pak turnusy, které jezdí pouze o prázdninách. Státní svátky a dny prázdnin jsou dle jednotlivých okresů uloženy v databázi. Kromě toho může každý turnus mít 3 druhy krátkodobé platnosti. ‚Nejede pouze' určuje dny, ve které turnus nejede nezávisle na ostatních pravidlech, ‚Jede pouze' značí, že ve všechny ostatní dny turnus nejede a ‚Jede také' může určit dny, kdy turnus jede i mimo standardní pravidla. Cílem tedy bylo vygenerovat bitovou mapu, podle které by se následně mohly dohledávat spoje. Co představuje na tomto projektu problém je to, že se musí počítat i s přespůlnočními spoji, pro které platí mnohem složitější pravidla. To se prakticky řeší tzv. alternativní bitovou mapou.

Dřívější pokusy jiných kolegů o implementaci generování bitových map v .NETu bohužel ztroskotaly. Byla tedy vyvinuta procedura v T-SQL, která danou problematiku řešila. Jak se ale v praxi ukázalo, procedura byla volána i několik set krát za minutu a rychlost procedury v T-SQL byla nedostatečná. Mým úkolem bylo napsat CLR proceduru v .NETu, která by se následně integrovala do databáze, a zoptimalizovat ji pro maximální výkon. Kromě toho musela procedura umět i vygenerovat na požádání DEBUG výstup, aby bylo možné dohledat chyby v zadání či programu samotném.

Nabízela se možnost práce s kolekcemi booleanů značících jednotlivé bity. Jelikož je ale v .NETu i boolean alokován jako 1 byte, rozhodl jsem se pro efektivnější volbu. Bity jsem zachytil do pole bezznaménkových long čísel (celé číslo alokované na 64 bitů) a do slovníku jsem si uložil ke každému dni pozici příslušného bitu. Slovník jsem zvolil, protože přístup k datům probíhá s konstantní složitostí nezávisle na počtu uložených prvků, tedy přesně to, co jsem potřeboval. K jednotlivým bitům jsem následně přistupoval pomocí bitových operací (>>, <<, &, |, ^, ~). Pravidla prázdnin, státních svátků apod. jsem rovněž znázornil poli bezznaménkových long čísel a jejich započítání jsem prováděl příslušnými bitovými operacemi na celá čísla. Tímto přístupem jsem tedy aplikoval v pár krocích pravidlo na 64 bitů zároveň. Kdybych oproti tomu používal např. kolekce booleanů, musel bych sekvenčně aplikovat pravidla na každý bit zvlášť, což by se na výkonu rozhodně podepsalo.

Obtížnější bylo vymyslet generování alternativní bitové mapy pro přespůlnoční turnusy. Alternativní bitová mapa obsahuje 1 tam, kde se turnus dojíždí ráno z předchozího dne. Aby však bylo možno zajistit návaznosti spojů, musí být na alternativní bitové mapy aplikována další pravidla typu „kdyby den před tím/poté za určitých podmínek turnus jel/nejel …“ jako např.:

- pokud je turnus sváteční, po půlnoci ze svátku se nedojíždí, pokud následující den není svátek
- alternativní bitovou mapu omezují krátkodobé platnosti stejně jako základní
- v případě že se mění krátkodobá platnost, alternativní bitová mapa má na první pozici jedničku tehdy, pokud by se bez aplikace platností den před tím turnus jel (pouze základní platnost 1-7, + NE!)
- …

Příklad práce s bity během generování alternativní bitové mapy:

```
// pokud bylo použito NEJEDE V, musí se vynulovat všechny bity, kde se nejede
if (jizdyNe != null)
{
    posunutaPravidla = ShiftOneBitLeft(jizdyNe);  // posunutí JEDE POUZE V o 1 bit
    for (int i = 0; i < jizdyVDen.Length; i++)
    {
        // získání zlomú z 0 na 1 a z 1 na 0 (funkce XOR: 0^1=1, 0^0=0, 1^1=0)
        zmeny[i] = (jizdyNe[i] ^ posunutaPravidla[i]) & omezeniSvatku[i];
       // pokud by den před změnou turnus podle zákl. platnosti jel, nastavit na 1
        jizdyAlternative[i] |= (posunutaZaklPlatnost[i] & zmeny[i]);
        jizdyAlternative[i] = ~jizdyAlternative[i];       // aplikování filtru
        jizdyAlternative[i] |= jizdyNe[i];                // -II-
        jizdyAlternative[i] = ~jizdyAlternative[i];       // -II-
    }
}
```

Podle posledního testování je CLR procedura v průměru 62 krát rychlejší než původní SQL procedura. Měření proběhlo na 8583 turnusech v době, kdy na serveru nebyl žádný jiný provoz. Průměrný čas běhu původní SQL procedury byl 62 ms, kdežto průměrná doba vykonání CLR procedury byla 1 ms.

4.3.3. Obecná komponenta pro vyhledávání v rozsáhlých datech

Moderní uživatel je dnes často zvyklý na to, že se mu během zadávání hledaného výrazu zobrazuje našeptávač. Ten vyhledává nejen od začátku slova, ale i uprostřed a poskytuje základní přehled o průběhu a správnosti hledání či zadávání. Není tedy divu, že požadavky zákazníků ohledně podobné funkcionality dorazily i do CID. Syncfusion obsahuje komponentu AutoComplete, která se dá napojit na libovolný TextBox, přiřadí se jí datový zdroj, nadefinují sloupce a o zbytek se komponenta postará sama. Proč tedy znovu vynalézat kolo, když už je jednou hotovo. Problém však nastal v případě, když jsme chtěli vyhledávat v rozsáhlejší kolekci dat. Například když jsme chtěli vyhledávat v seznamu firem, které se v běžné databázi zákazníka mohou počítat ve statisících, stala se komponenta od Syncfusionu naprosto nepoužitelnou. Pouze naplnění komponenty daty trvalo pro cca 200.000 záznamů téměř půl minuty a následné vyhledávání v datech zablokovalo celou aplikaci na

přibližně 10 sekund po každém napsaném či smazaném znaku. Takovéto chování bylo samozřejmě neakceptovatelné, a proto jsem byl pověřen úkolem pokusit se vyvinout něco, co by zachovávalo funkcionalitu, avšak s minimální časovou složitostí.

Problém Syncfusionu se ukázal být především v tom, že si uvnitř data kopíroval do svých struktur. Data pak fyzicky byla přítomna dvakrát - jednou v samotném zdroji a jednou ve struktuře komponenty. Podobné chování má defaultně např. i klasický ComboBox, který je rovněž pro větší množství záznamů nepoužitelný. V průběhu prvních pokusů jsem zjistil, kterým směrem se vydat, až ve chvíli kdy jsem objevil klasický ListView, ale s možností VirtualMode = true. Společně s implementací obsluhy události RetrieveVirtualItem na získání ListItem podle indexu toto nesrovnatelně zlepšilo výkon při načítání dat pro zobrazení. Od této chvíle už byla data pouze na jednom – zdrojovém – místě a ListView vždy zobrazoval pouze jejich interpretaci a to navíc jen těch prvků, které byly zrovna viditelné. Tato vlastnost je dost klíčová, protože pro představu, když si uvědomíme, že v modulu může být podobných komponent více, pro ukázku dejme tomu 4, a každá komponenta by si k sobě data zkopírovala, tak při datovém zdroji s 200.000 záznamy, který samotný v paměti zabere něco málo přes 100 MB, by paměťová náročnost zbytečně vzrostla na 0,5 GB.

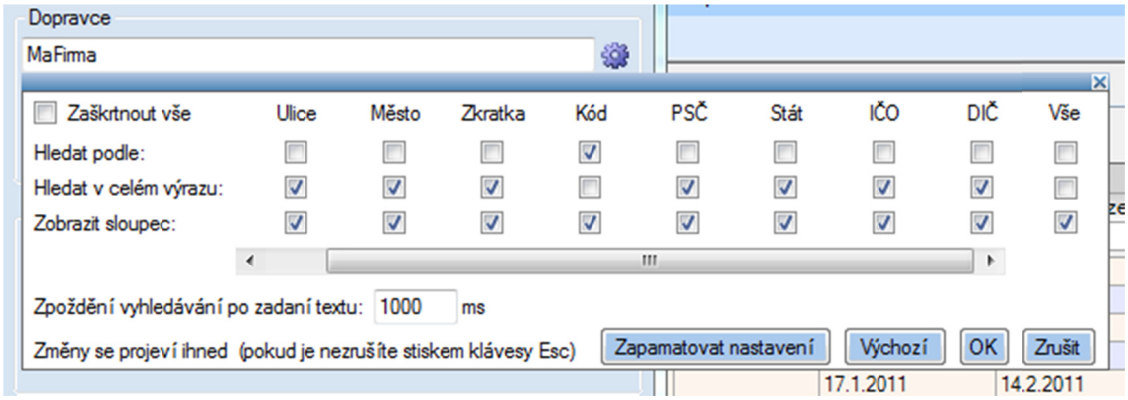
Podařilo se tedy snížit paměťovou náročnost na pouhou velikost datového zdroje. Nicméně při velkém počtu záznamů může být i velikost samotné kolekce problém. Naše aplikace v provozu běží z pravidla na terminálovém serveru, a když by si každý uživatel spustil svůj modul, tak by i pouhé načtení samotných dat mohlo vyústit v pád aplikace kvůli nedostatku paměti. Naskytly se v podstatě dvě možnosti řešení. Buď bude datový zdroj načten pouze jednou a pro všechny moduly bude sdílený, nebo se bude vyhledávat přímo v databázi.

První možnost - sdílený číselník - by umožňoval minimalizovat komunikaci s databází po síti a tedy výrazně zkrátit dobu načítání dat. Bylo by ale třeba řešit několik problémů. V paměti by se musel držet kompletně celý požadovaný obsah databáze, protože každý modul může pracovat s různými podmnožinami záznamů, přičemž se mohou podmnožiny prolínat. Pokud by se nasdílel omezený seznam záznamů pro jeden modul, druhý modul by už neměl všechna data, která potřebuje, a stejně by musel opět přistupovat k databázi. Dalším problémem je aktualizace. Pokud by někdo přidal či upravil nějaké záznamy v databázi, změny by se musely promítnout i do sdílené kolekce. Musíme také vzít v potaz, že aplikace může běžet třeba měsíc v kuse a muselo by se řešit i uvolňování dat z paměti, nejsou-li dlouhodobě používána.

Naopak pokud bychom pracovali přímo s databází, v paměti bychom měli vždy pouze záznamy, které jsou pro nás smysluplné, avšak za cenu časté komunikace s databází. V praxi se následně ukázalo, že pro velké počty záznamů je tento přístup mnohem rychlejší. Konec konců není divu, protože databáze jsou pro vyhledávání v rozsáhlých strukturách optimalizovány.

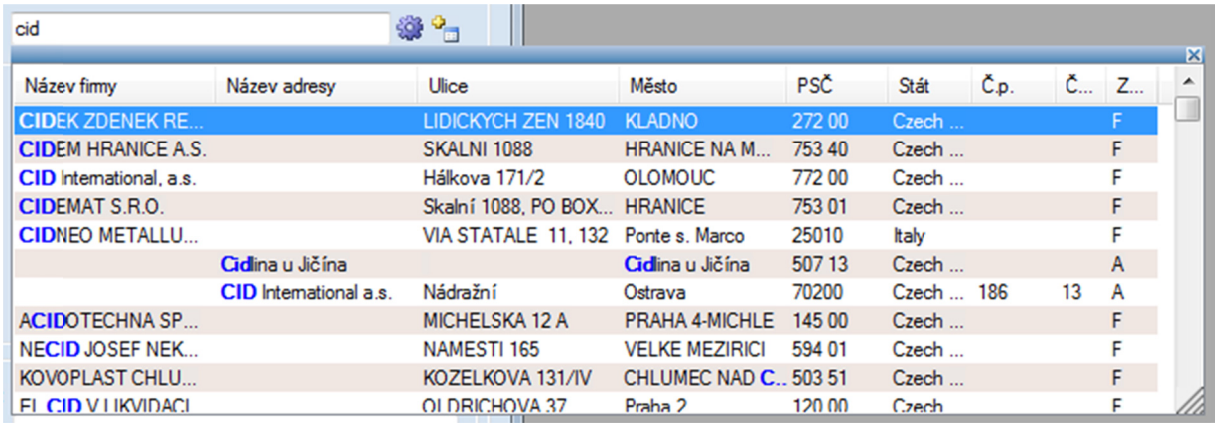
Pokud jde tedy o komponentu, kterou jsem měl vytvořit, měla umět obojí. Podle nastavení musela umět vyhledávat v předaném seznamu entit, anebo nechat hledání na databázi a pouze interpretovat výsledky. Komponenta musela být maximálně přizpůsobitelná. Na požádání musí prvek umět vygenerovat i obecný konfigurační dialog pro libovolný počet sloupců. Pro každý sloupec se pak dá nastavit, zda se má sloupec zobrazovat ve výsledcích, zda se v něm má vyhledávat a jestli se má ve sloupci případně vyhledávat od začátku slova, nebo v celém textu. Aby se zabránilo zbytečnému

vyhledávání okamžitě po každém stisku klávesy, dá se nastavit i zpoždění, po jehož uplynutí se vyhledávání spustí. V praxi to znamená, že dokud uživatel píše, nic se nevyhledává. Jakmile přestane psát, čeká se nastavenou dobu, jestli uživatel nezačne opět psát. Pokud ne, teprve poté se spustí vyhledávání. Toto opatření reaguje na problém, kdy komponenta od Syncfusion po zadání či smazání znaku zamrzla, dokud neobdržela výsledky hledání.



Obrázek 2: CIDControlAutoComplete - dialog nastavení v CIDControlFirmaGrid

Samotné vyhledávání nesmí v žádném případě zablokovat celé GUI. To znamená, že vyhledávání musí běžet na pozadí v jiném vlákně. Pokud uživatel začne opět psát, běžící vyhledávání se okamžitě ukončí, aby následně mohlo začít nové. Pokud se vyhledává v předaném seznamu entit, který obsahuje více než 10.000 záznamů, je vyhledávání provedeno ve 4 vláknech, což na více jádrových procesorech proces urychlí. Aby bylo vidět, že komponenta na pozadí pracuje, je v ní zakomponován ProgressBar, který se při každé činnosti automaticky zobrazí a signalizuje průběh procesu. Protože nemá smysl zobrazovat prázdné okénko v případě, kdy nebyly nalezeny žádné výsledky, indikuje komponenta nenalezení odpovídajících záznamů změnou barvy textu ve vyhledávacím poli. Nalezené výsledky jsou setříděny tak, že první jsou zobrazeny záznamy se shodou od začátku slova a až po nich záznamy se shodou uprostřed slova. Čím víc je sloupec vlevo, tím má větší prioritu. Pro lepší přehled ve vyhledaných výrazech umí komponenta zvýraznit všechny výskyty hledaného textu tučně a rovněž je kontrolována pozice zobrazení dialogů nastavení a nalezených výsledků tak, aby byly vždy zobrazeny celé (např. aby nebyla část skryta za okrajem obrazovky).



Obrázek 3: CIDControlAutoComplete - zobrazení nalezených výsledků

Komponenta by měla postupně částečně nahradit klasický ComboBox. To znamená, že musí umožňovat výběr z vyhledaných prvků, ale také musí umět poskytnout samotný vybraný text. Komponenta může a nemusí pracovat s primárními klíči záznamů. Obsah sloupců vybraného řádku je pak k dispozici v setříděném slovníku (ten umožňuje přístup k hodnotám jak přes index, tak přes textový klíč) SelectedRowValues a v případě vyhledávání v seznamu entit i v objektu SelectedEntity. Případný primární klíč pak lze najít v propertě SelectedValue.

Aby uživatel nemusel nastavovat parametry vyhledávání neustále znovu a znovu při každém spuštění modulu, umí komponenta na základě jedinečných informací uživatelské nastavení uložit do databáze, odkud si jej následně může obnovit.

V neposlední řadě byl kladen důraz i na ovládání celé komponenty pomocí klávesnice.

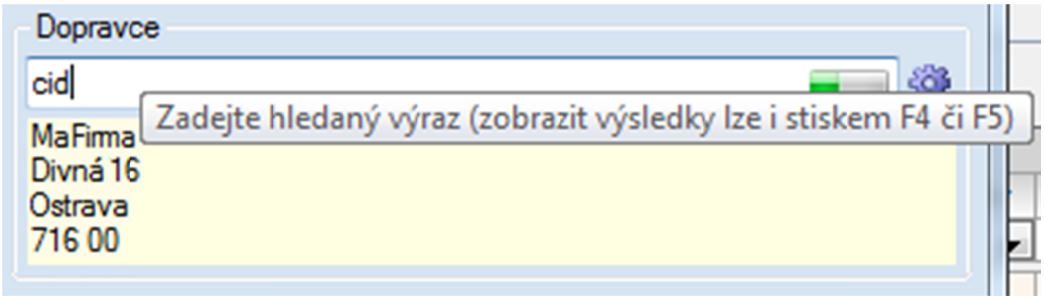
Klávesová zkratka	Funkce
F4, F5	Spuštění vyhledávání, pokud ještě samo na základě nastaveného zpoždění nezačalo, nebo zobrazení/skrytí vyhledaných výsledků
Esc	Zastavení vyhledávání, skrytí okénka s výsledky, nebo vrácení provedených změn v nastavení
Ctrl+Mezerník	Zobrazení dialogu s nastavením parametrů vyhledávání
Ctrl+D	Obnovit výchozí nastavení parametrů vyhledávání
Ctrl+S	Uložit nastavení parametrů vyhledávání pro daného uživatele
Enter	Potvrdit změny v nastavení a zavřít dialog
Alt+Ctrl+Shift+D	DEBUG režim – popisy všech chyb, které případně nastanou, jsou zobrazeny v MessageBoxech

Tabulka 3: Klávesové zkratky CIDControlAutoComplete

4.3.4. CIDControls

CIDControls jsou určité atomické prvky, které jsou využívány častěji než jednou. Ani já jsem se jejich vývoji nevyhnul. Zde jsou některé z nich:

CIDControlFirmaGrid – control pro vyhledávání ve firmách a zobrazení detailů vybrané firmy. Původně vznikl kvůli problému se Syncfusion.AutoComplete, který byl pro velké množství firem v databázi neskutečně pomalý. Po úspěšném vzniku tohoto prvku přišel impulz pro vytvoření vlastní CIDControlAutoComplete komponenty, kterou dnes již využívá.



Obrázek 4: CIDControlFirmaGrid - probíhající hledání

CIDControlAutoComplete – viz *4.3.3. Obecná komponenta pro vyhledávání v rozsáhlých datech*.

CIDControlAdresa, **CIDControlAdresaText** a **CIDControlAdresaGrid** – trojice controlů, které mají na starost znázornění informací o adresách a práci s nimi. CIDControlAdresa slouží k vyhledávání různých typů adres v databázi, k jejich modifikaci a k následnému uložení. CIDControlAdresaText zobrazuje detaily jedné konkrétní adresy, kterou umožňuje změnit skrze CIDControlAdresa. Poslední z trojice je CIDControlAdresaGrid, který zobrazuje adresy v tabulce. Tyto controly jsem převzal za účelem optimalizace jejich výkonu a měl jsem minimalizovat paměťové nároky a dobu inicializace. CIDControlAdresa již také využívá CIDControlAutoComplete pro dohledávání adres a měst. Zlepšení, kterých jsem u těchto controlů dosáhl, jsou obrovská. Pokud se před mými zásahy control načítal téměř půl minuty a zabíral přes 100 MB paměti, tak nyní se načte za přibližně 300 ms a zabírá pouhých pár kB. Ukázka viz *Příloha 3: Ukázka CIDControlAdresa*.

4.3.5. Moduly

Jako každý programátor ve firmě jsem měl i já na starosti vývoj některých modulů. Pracoval jsem tedy např. na modulech pro administraci skladovatele či ukladatele, nebo na modulu dopravní prostředek, který byl mým vůbec prvním projektem po mém nástupu.

Dopravní prostředek (_scbDoprProstredek) slouží k administraci informací o tahačích, návěsech či autobusech. V modulu lze zadat např. registrační značku, evidenční a technická data, řidiče, kteří s vozem jezdí, vybavení auta, různá osvědčení, nehodové události, evidence náhradních dílů či platebních karet, umístění reklam na vozidla atd. Ukázka viz *Příloha 2: Ukázka modulu _scbDoprProstredek*.

Administrace skladovatele (_scbAdmSkladovatel) a **ukladatele** (_scbAdmUkladatel) umožňují zákazníkovi, aby si sám nastavil číselníky a nejrůznější nastavení, která jsou v aplikaci zapotřebí a která by jinak musela nastavovat technická podpora přímo v databázi. Pro představu lze nastavit např. typy a struktury EAN kódů, algoritmy pro zobrazování dat, parametry příjemek a výdejek, skladové karty aj.

Jednoduchá zásilka (_scbZasilKalk) je modul, jehož cílem je připravit data v dialogu a po kliknutí na tlačítko Kalkulovat se musí data předzpracovat v databázi, následně je zavolán modul _coKalk napsaný v C++ a podle jeho výsledků a potvrzení uživatele se případné změny musí buď započítat, nebo zrušit.

Plachta pro kamiony (_anPlaPtr) je modul, který jsem převzal po 2 dalších programátorech. Je napsána v .NET 1.1 a protože můj poslední předchůdce nebyl zrovna logicky zdatný, obsahuje spoustu zbytečných chyb. Zdrojový kód je téměř neuspořádaný a většinou bez komentářů, takže je velmi obtížně se v něm orientovat. Modul slouží k přehlednému zobrazení a administraci jízd kamionů, jejich zásilek, naplnění apod.

4.3.6. Import a Export dat

Import avíz pro OKD (_anExpImpTXT) je automatický import dat z textových souborů do databáze, přičemž formát obsahu se může mírně lišit a může být zaslán v různých jazycích.

Import informací o zásilkách nákladních vlaků (_anIsopcXml) je automatický import a zapracování dat zaslaných v XML souboru do databáze. Na tomto projektu bylo složité pouze to, že importovaný XML formát (zaslaný jinou firmou) byl navržen naprosto nevhodně. Všechny tagy byly v rootu a odkazovaly na sebe několika nejednoznačnými ID (jednoznačná byla pouze jejich úplná kombinace). Originální strukturu dat se podařilo sestavit až po několika násobném procházení celého obsahu v cyklech (nešlo předpokládat, že se někde v dokumentu již nějaký prvek nevyskytuje).

4.4. Optimalizace výkonu

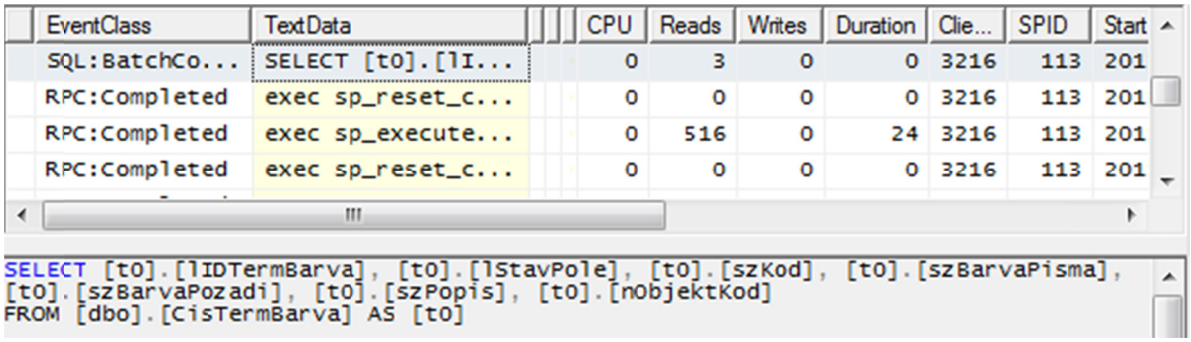
Rychlost a nenáročnost aplikace je velmi důležitým faktorem, který při rozhodnutí, zda si IS koupit, či nikoli, může sehrát klíčovou roli. Pokud bychom měli krásný funkčně dokonalý systém, ve kterém se ale každý modul načítá půl minuty a zabere nám polovinu RAM paměti, asi bychom z něj nadšení nebyli. Proto je třeba monitorovat nedokonalosti a snažit se neustále zdokonalovat kód v případě, že byl nějaký nedostatek nalezen. Velkým pomocníkem ve zjišťování slabých míst jsou nástroje jako např. MS SQL Profiler, EQATEC Profiler či nástroj Performance Explorer zabudovaný přímo ve Visual Studiu 2010.

4.4.1. MS SQL Profiler

„SQL Server Profiler je grafický nástroj, který slouží jako interface k SQL Trace Application Programming Interface (API). Profiler umožňuje definovat události SQL Serveru, jejichž informace chcete zachytit.“ [5] Jinými slovy řečeno slouží k monitorování veškeré komunikace s databází MS SQL Server. Analyzující osoba se připojí k serveru, na kterém DB běží, nastaví si, které události ji zajímají, nastaví požadované filtry a spustí trace. V zachyceném provozu je poté třeba se zaměřit především na následující problémy:

Objekt sledování:	Kdy může být problém:	Možná reakce:
Četnost dotazů	Pokud se volá nějaká metoda vícekrát nebo si programátor výsledky poprvé neuložil	Než dvakrát přistupovat do databáze, je lepší si výsledky poprvé uložit a následně použít lokálně načtená data. Zkontrolovat, jestli se některá z metod nevolá zbytečně vícekrát
Doba vykonání dotazu a zatížení serveru	Pokud dotaz není napsán efektivně	Pokusit se o sestavení optimálnějšího SQL dotazu, při JOIN operacích zvážit vytvoření VIEW
Množství vrácených záznamů	Při přístupu k DB přes ORM můžou dotazy vracet zbytečně hodně záznamů	Pokusit se napsat efektivnější metody pro práci s daty, nebo přehodnotit použití ORM

Tabulka 4: Ladění aplikací pomocí MS SQL Profileru

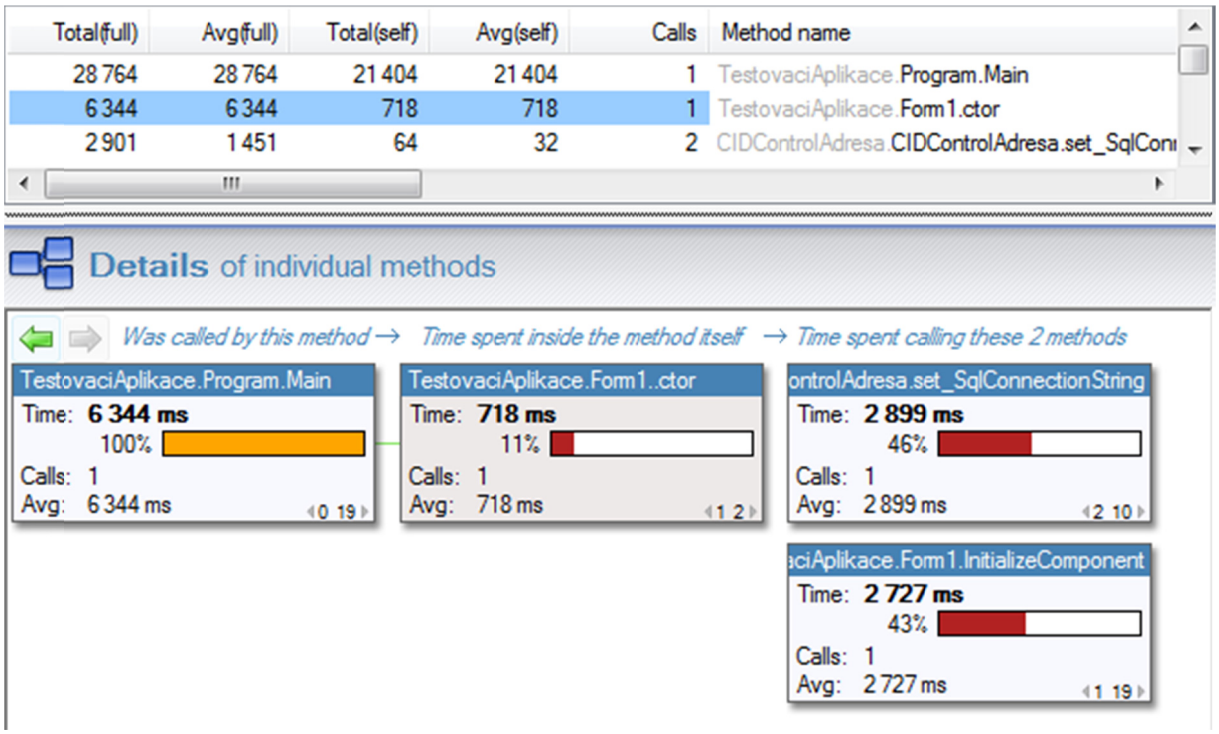


Obrázek 5: MS SQL Profiler - ukázka zachytávání provozu

4.4.2. EQATEC Profiler

EQATEC Profiler je nástroj, který umožňuje získat časovou analýzu vykonávání jednotlivých metod v programu. Je to velice užitečný nástroj, pokud máte pomalou aplikaci, chcete ji zrychlit a nemáte tušení, co konkrétně ji zpomaluje. Tento nástroj umožňuje analyzovat časové složitosti jednotlivých metod a snadno dohledat problém.

V programu načtete dll knihovny, které chcete monitorovat, přeložíte je znovu přímo v programu a testovanou aplikaci následně spustíte. Jakmile v aplikaci provedete všechny úkony, které chcete monitorovat, klasicky ji ukončíte, čímž se vrátíte do EQATEC Profileru, kde se vygeneruje tzv. snapshot. Snapshoty si následně můžete nechat zobrazit v přehledném grafickém znázornění a klikáním si zobrazit problémová místa viz *Obrázek 6: EQATEC Profiler - příklad zobrazení časové náročnosti metod*.



Obrázek 6: EQATEC Profiler - příklad zobrazení časové náročnosti metod

4.4.3. Performance explorer ve Visual Studiu 2010

Pokud chcete podrobnou analýzu náročnosti nějakého programu a to přímo se závislostí na zdrojový kód, je Performance explorer jednou z ideálních možností. Umožňuje různé úrovně monitorování, přičemž dokáže vytvořit komplexní analýzu kódu a znázornit výsledky ve stromech či grafech. Jednoduše tak lze zjistit, který konkrétní řádek ve zdrojovém kódu dělá problémy a dokonce umí Performance explorer i navrhnout odhadované řešení problému. Například pokud zjistí časté spojování řetězců operátorem ‚+', navrhne použití StringBuilderu apod.

4.5. Moje práce v praxi

Pro klasickou firmu bývá každý projekt, který se nedostane do praxe, pouhým plýtváním prostředků. U CID tomu není jinak. Všechny části, na kterých jsem pracoval, již v tuto dobu používají zákazníci. Generování bitových map je hojně využíváno při dohledávání spojů v dopravě, modul Dopravní prostředek je klíčový pro evidenci tahačů, návěsů a autobusů, moduly Administrace skladovatele a Administrace ukladatele zase pomáhají zákazníkům ušetřit finance a čas za technickou podporu. Importní projekty se starají o konzistenci dat napříč informačními systémy.

Patrně největší můj přínos pro společnost CID a pro pohodlí zákazníků je ale komponenta CIDControlAutoComplete, která výrazně usnadňuje dohledávání jakýchkoli záznamů nezávisle na množství dat, se kterými pracuje, a svou univerzálností je schopna efektivně nahradit například klasické ComboBoxy či dokonce celé moduly, které byly pro konkrétní vyhledávání vyvinuty dříve.

5. Závěr

Cílem mé odborné praxe bylo získat praktické zkušenosti s vývojem informačního systému na základě Smart Client Software Factory, přispět společnosti CID International a.s. svou prací k rozšíření stávajícího IS, ať už jde o moduly, controly či komponenty a splnit všechny podmínky potřebné pro úspěšnou obhajobu praxe.

Po celou dobu mé práce jsem kladl důraz na efektivitu a maximální spolehlivost mnou vytvářených kódů a snažil jsem se, aby uživatelská rozhraní, která jsem mohl ovlivnit, byla pokud možno co nejlépe uživatelsky přívětivá.

Můj názor je, že se za svou práci nemusím stydět, naopak si myslím, že např. s generováním bitových map pomocí bitových operací či s CIDControlAutoComplete komponentou by spousta stávajících programátorů měla dost velké problémy.

S firmou CID International a.s. hodlám spolupracovat i nadále, o čemž svědčí i můj rozpracovaný projekt vytvoření paměťového objektu pro aplikační server, který je dlouhodobějšího rázu. Budu dělat vše, co je v mých možnostech, abych se za svou práci nemusel stydět ani v budoucnu.

Pro mne byla má odborná praxe velkým přínosem. Seznámil jsem se s různými fázemi vývoje IS a s různými způsoby vedení týmů lidí (zajímavá je např. technika SCRUM). Aktivně jsem se účastnil některých klíčových porad a absolvoval jsem pár interních školení (např. ohledně práce

s programy MS SQL Profiler, Performance Explorer, aj.). Poznal jsem způsob vytváření IS na základě SCSF a prakticky jsem tímto způsobem i vyvíjel. Vytvářel jsem moduly, controly, komponenty, servicy a pracoval jsem aktivně s databází MS SQL Server. Zvyknul jsem si na odpovědnost za svou práci a veškerá má práce byla nasazena u zákazníků. Převzal jsem i pár již dříve nasazených projektů po jiných bývalých programátorech, které byly napsány třeba i v .NET 1.1, a opravoval jsem v nich chyby. Dostal jsem na starost zoptimalizování některých starších controlů a doplnění funkčností nových. Naučil jsem se pracovat s verzováním, psal jsem dokumentace k vytvořeným projektům a to jak do externího souboru, tak v rámci komentářů ve zdrojových kódech …

Pokud mám uvést vědomosti, které mi chybí a které bych uvítal, byla by to určitě lepší znalost syntaxe T-SQL, která se od mě známé PL-SQL docela dost liší, a celkově bych uvítal více znalostí a zkušeností s databází MS SQL Server. Pro vývoj aplikačního serveru bych dále potřeboval znalosti ohledně WCF, distribuovaného kešování, mapování objektů, různých frameworků ohledně vývoje aplikačních serverů apod.

Cílem této práce bylo představit společnost, se kterou jsem spolupracoval, decentně seznámit se softwarovými prostředky, se kterými jsem pracoval, ale především poskytnout přibližný přehled o mnou dokončených projektech a výsledcích, jichž jsem v nich dosáhl. Z mého pohledu jsem tedy zadání bakalářské práce splnil. Kdybych se měl rozhodnout znovu, jestli odbornou praxi, nebo samostatnou bakalářskou práci, rozhodl bych se bez váhání opět pro odbornou praxi už jenom z důvodu, že jsem přesvědčen o tom, že většinu výše popsaných zkušeností bych ve škole neměl šanci získat.

6. Seznam použité literatury

[1] ABRAMS, Brad. *MSDN* [online]. 26.1.2005 [cit. 2011-04-17]. Internal Coding Guidelines. Dostupné z WWW: <http://blogs.msdn.com/b/brada/archive/2005/01/26/361363.aspx>.

[2] BERAN, Jiří. *Využití technologie Microsoft Software Factories pro zefektivnění vývoje modulárních spedičních a logistických aplikací pro operační systém MS Windows ve vývojovém prostředí MS Visual Studia 2008*. Ostrava, 2010. 30 s. Bakalářská práce. VŠB-TU Ostrava.

[3] *CID International a.s.* [online]. 2008 [cit. 2011-04-17]. Home. Dostupné z WWW: <http://www.cid.cz>.

[4] *CID International a.s.* [online]. 2008 [cit. 2011-04-17]. Microsoft Gold Certified Partner. Dostupné z WWW: <http://www.cid.cz/clanek.m?Id=145>.

[5] HOTEK, Mike. *Microsoft® SQL Server® 2008 Step by Step*. Redmond (Washington) : Microsoft Press, 2008. 800 s. ISBN 0-7356-2604-9, ISBN 9780735626027.

[6] PLATT, David S. *Programming Microsoft® Composite UI Application Block and Smart Client Software Factory*. Redmond (Washington) : Microsoft Press, 2007. 224 s. ISBN 0-7356-2414-3, ISBN 9780735624146.

7. Seznam tabulek

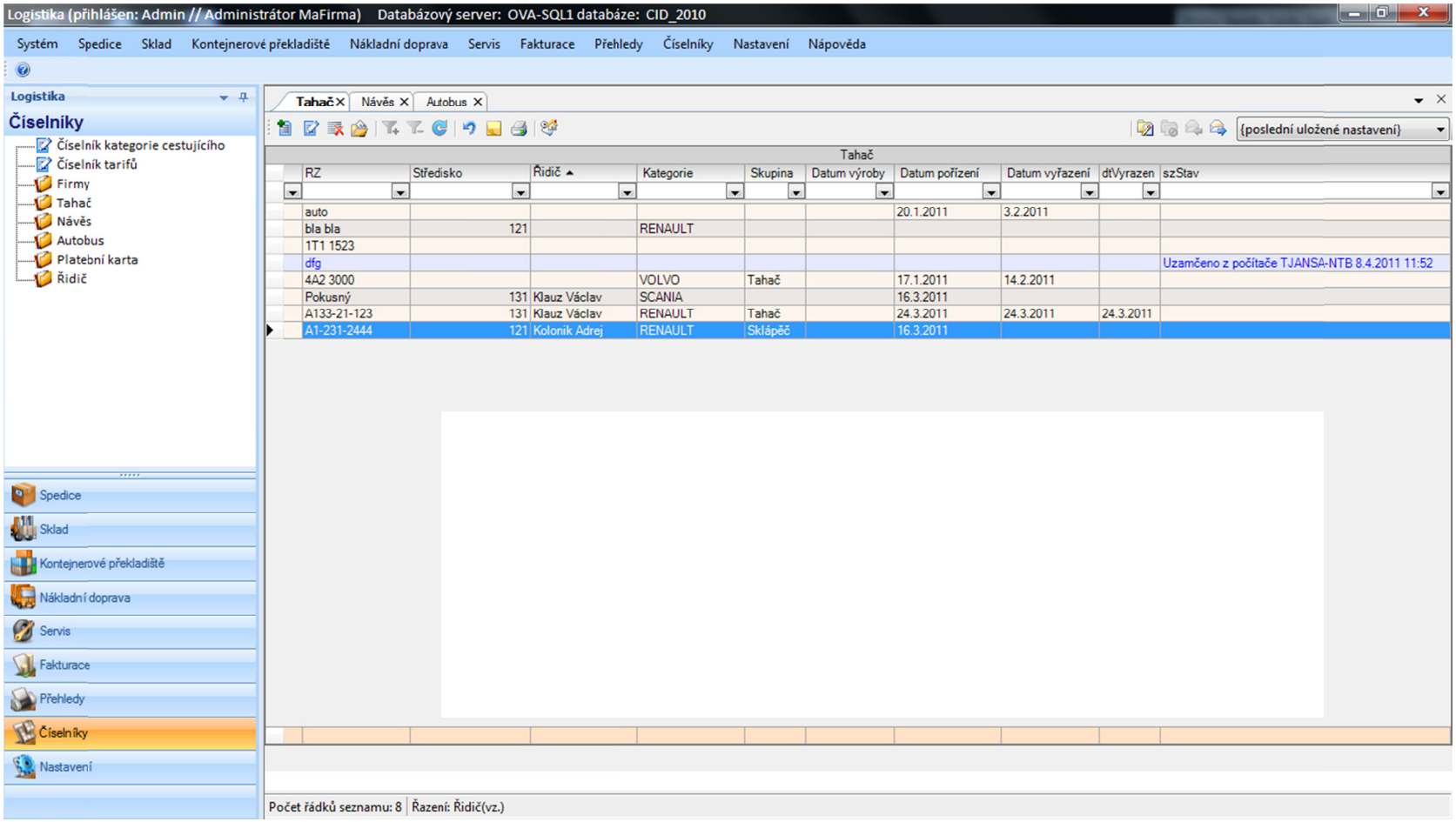
8. Seznam obrázků

9. Seznam příloh

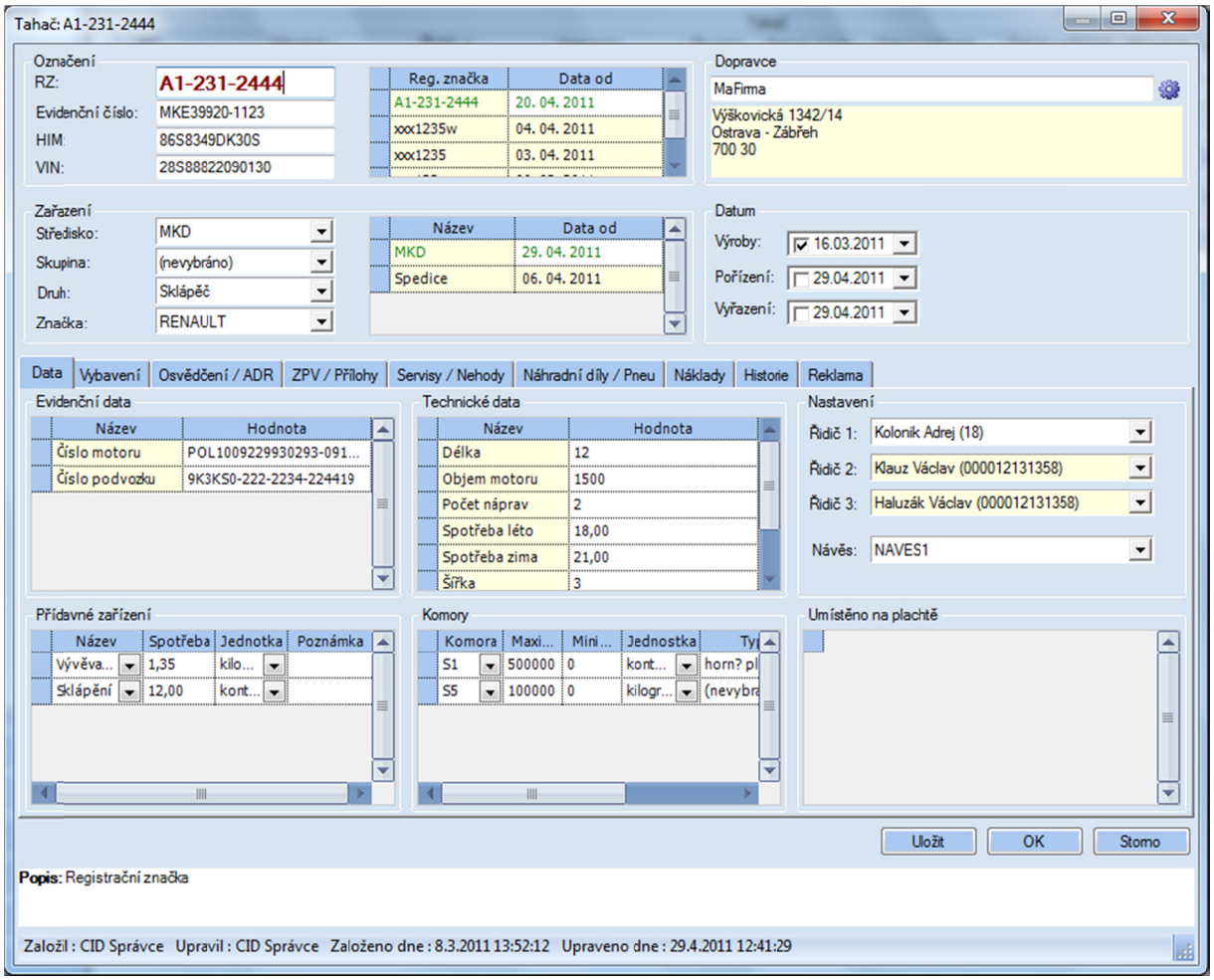
Příloha 1: Ukázka IS Lori
Příloha 2: Ukázka modulu _scbDoprProstredek
Příloha 3: Ukázka CIDControlAdresa
Příloha 4: Ukázka #REGION struktury kódu v CidControlAutoComplete
Příloha 5: Ukázka práce se Source Safe (plugin TFS ve VS2010)

Přílohy

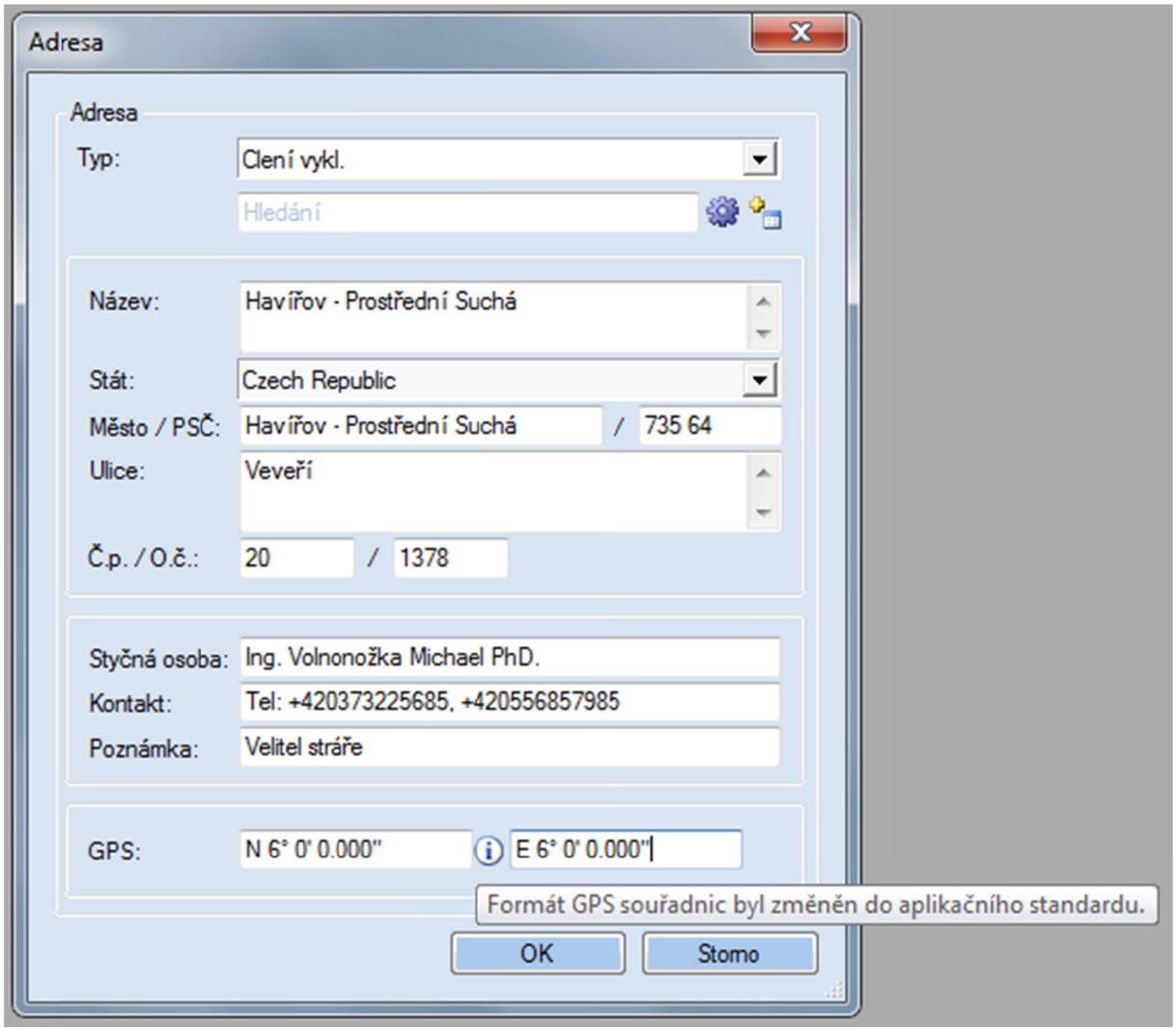
Příloha 1: Ukázka IS Lori



Příloha 2: Ukázka modulu _scbDoprProstredek



Příloha 3: Ukázka CIDControlAdresa



Příloha 4: Ukázka #REGION struktury kódu v CidControlAutoComplete

Příloha 5: Ukázka práce se Source Safe (plugin TFS ve VS2010)